Project Milestone 1 (Commonsense Validation)

Aarshee Mishra aarsheemishr@
Anupam Yadav anupamyadav@
Sangeetha Balasubramanian sangeethabal@
Sumeha Kashyap sumehakashya@

1 Introduction

Natural Language sentences can often be syntactically and grammatically correct but may not make any sense. Common sense reasoning is one of the main bottle necks in machine intelligence. Research on common sense understanding systems has received increased attention. There are quite a few benchmark tasks and datasets that evaluate a system's ability to infer commonsense knowledge in order to reason and understand natural language text. In our work we analyze some of the existing approaches and benchmarks to differentiate natural language statements that make sense from those that do not make sense. We will then try to introduce a novel way of differentiating statements on the basis of commonsense.

2 Your dataset

For our first task, we choose from two natural language statements with similar wordings which one makes sense and which one does not make sense. For our second task, we need to find the key reason why a given statement does not make sense. For each of the subtasks, we have 2,021 samples. Task 1 has 2,021 against common-sense sentences, 2,021 correct sentences. Task 2 has 2,021 true reasons and 4,042 confusing reasons. The entire data has been annotated and is available online [1]. The dataset has been released as part of the SemEval 2020 contest and hence we have an easy access to it. This dataset (Wang et al., 2019) is appropriate for our tasks which is used to directly test whether a system can differentiate natural language statements that make sense from those that do not make sense.

Formally, each instance in our dataset is composed of 5 sentences: {s1, s2, r1, r2, r3}. s1 and s2 are two similar sentences which in the same syntactic structure and differ by only few words, but only one of them makes sense while the other does not. These will be used on our first subtask called Sen-Making, which requires the model to identify which one of the statements is valid. For the invalid sentence, we have three optional reasons r1, r2 and r3 to explain why the sentence is invalid.

For the first task , the data is organized in a pairwise manner. For every sentence that does not make sense, there is a sentence that does make sense. Some examples of sentences that don't make sense are: 'He poured orange juice on his cereal.' , 'He drinks apple.' ,'Jeff ran 100,000 miles today.'. While the corresponding sentences that make sense are: 'He poured milk on his cereal.' , 'He drinks milk.', 'Jeff ran a mile today.'

3 Baselines

We have used naive-bayes classifier as our baseline approach. We used whitespace tokenization to create a bag-of-unigrams representation for the input text. We chose accuracy as our evaluation metric based on whether our model has classified sentences correctly or not. We used 83:17 train/test split. That translates to 10000 sentence pairs for training and 2021 for testing. Out of 10000 training examples we plan to reserve 20% for validation. Using α as 0.2 we observed accuracy of 63.14% on the test data.

4 Error analysis

Some examples which our baseline approach misclassifies:

- I put my desktop into my suitcase before departure

[1] https://github.com/wangcunxiang/Sen-Making-and-Explanation

	Positive Example Statistics	Negative Example Statistics
Total Number of Sentences	10,000	10,000
Avg Number of Characters	38.75	38.84
Avg Number of Words	7.66	7.68
Avg Number of Common Words in every sentence pair	6.20	
Avg Number of Different Words in every sentence pair	1.46	

Table 1: Datatset Statistics

- this bucket can hold one gallon of water
- I put the rubbish into a trash can
- a salad usually contains grass
- computers are everywhere in a forest

Since our proposed approach does not make the naive assumption of unigram probability we hope to get much better results.

5 Your approach

As the first step, we analyzed the data for our first task. We loaded the pre-trained Bert tokenizer and used it to tokenize the sequences in our data with the 'bert-base-uncased' pre-trained weights. This tokenizes our data (splits sequences into words) and converts all characters to lower case. We then encoded it into a sequence of hidden-states using the BertForSequenceClassification model. For our next step, we tried to calculate a score per input sentence. While calculating the scores, we notice random values. We realised this is not possible because Bert is not a language model (we can't get sentence probability) and also that we haven't trained Bert from scratch.

Our implementation is in progress. We are using pytorch-transformers and pytorch-pretrained-bert. We are running our experiments on Google Colab.

6 Timeline for the rest of the project

1. We will train a fully connected layer using the Bert embeddings (Oct 25 - Nov 17)
2. Fine tune Bert to our dataset and check its performance (Nov 22 - Dec 6)
3. Incorporate external knowledge from various knowledge bases like ConceptNet to word embeddings (if time permits)
4. Work on final report and presentation (Dec 7 - Dec 21)

References

Wang, C., Liang, S., Zhang, Y., Li, X., and Gao, T. (2019). Does it make sense? and why? a pilot study for sense making and explanation. *arXiv preprint arXiv:1906.00363*.